TP 1 – Réécriture d'historique, fusion explicite des branches associées à des exigences, introduction au github-flow et pull request

Exercice 1 – Mise en place du projet « login-generator »

1. Créez un projet (public ou privé) sur GitHub nommé « login-generator ». Ce projet servira de fil directeur aux 3 TP de GeCo. Il est donc conseillé de finir complètement chaque feuille de TP avant de commencer la feuille suivante.
2. Créez sur votre poste de travail un projet Maven en utilisant l'archétype « maven-archetype-quickstart » avec les caractéristiques suivantes :
   groupId : geco **;** artifactId : login-generator **;** package : geco
   (vous choisirez « login-generator » comme nom de projet IntelliJ)
3. Votre projet utilisera JUnit en version 4.12 pour la création des tests unitaires. Modifiez le fichier pom.xml de votre projet pour exprimer cette dépendance.
4. Votre projet utilisera les plugins Cobertura (et jxr) pour mesurer la couverture du code par les tests sur votre projet. Modifiez le fichier pom.xml de votre projet pour pouvoir générer le rapport Cobertura.
5. Faites en sorte que votre projet soit supervisé par Git. On s'attachera à vérifier que le dossier « .git » soit bien à la racine de votre projet.
6. Faites en sorte que les dossiers « target », « .idea » et les fichiers « *.iml » soient ignorés par Git.
7. Remplacez le dossier « src » à la racine de votre projet par le dossier fourni sur Moodle.
8. Commitez les fichiers de votre projet et poussez-les sur GitHub (après avoir configuré le remote).

Exercice 2 – Spécification des exigences

Sur l'espace GitHub de votre projet, créez et assignez-vous les 2 « issues » suivantes :
#1 → Couvrir LoginService à 100 % par des tests unitaires
#2 → Couvrir LoginGenerator à 100 % par des tests unitaires

Ex. 3 – Couverture de LoginService par les tests, branches et merge no-fast-forward

Le but est de couvrir LoginService par les tests dans une branche dédiée, mais en supposant que *vous avez oublié au départ de commiter dans la bonne branche* (ceci afin de vous entraîner à utiliser les commandes vues en cours pour « jongler » entre les branches).

1. Dans la branche master, créez la classe de test unitaire qui permettra de tester la classe LoginService et supprimez la classe AppTest, maintenant superflue. Commitez ces 2 changements (sans effectuer de push).
2. Le ou les commits que vous venez de faire auraient dû être faits dans une branche dédiée testsLoginService associée à l'exigence #1.
   En utilisant le bon enchaînement de commandes, créez la branche manquante avec le(s) commit(s) considéré(s), rétablissez l'état de la branche master et basculez sur la branche testsLoginService.
   Enfin, vérifiez l'état de l'historique avec « gitk --all ».
   Vous pouvez vous appuyer sur le cours DéQo/GeCo et/ou sur la documentation de Git (git help -w branch, git help -w reset, git help -w checkout)
3. Écrivez les tests unitaires permettant de couvrir le code de la classe LoginService.
4. Contrôlez la couverture avec « mvn clean test site » et commitez votre code.
5. Fusionnez votre branche dans master *en faisant un merge non-fast-forward, en choisissant un message de commit contenant le mot-clé* « close #1 ».
   Pour faire cela dans IntelliJ (après être revenu dans la branche master), utiliser le menu VCS > Git > Merge Changes… et cocher « no fast forward ».

6. Contrôlez la forme de l'historique obtenu (avec « gitk --all » en ligne de commande ou avec l'onglet « Log » d'IntelliJ) : la séparation puis la fusion des deux branches doit être clairement identifiable.
7. Supprimez la branche locale testsLoginService et poussez la branche master. Vérifiez que l'Issue #1 a été automatiquement fermée par l'intégration dans master du commit avec le message « close #1 ».

Exercice 4 – Couverture par les tests et correction de LoginGenerator, réécriture d'historique et introduction au github-flow et pull request

Dans cet exercice vous allez maintenant couvrir la classe LoginGenerator par les tests, puis corriger un bug repéré durant cette phase d'élaboration des tests. Vous effectuerez ensuite un rebase --interactive (réécriture d'historique) pour nettoyer votre historique local avant de pousser votre branche locale, et enfin vous ouvrirez un Pull Request à l'intérieur de votre repository GitHub pour permettre la revue de code et la fusion dans master.

1. Créez une branche (en ligne de commande ou via une « tâche IntelliJ ») associée à l'exigence #2, nommée testsLoginGenerator et dérivant de la branche master. Basculez sur cette branche.
2. Créez la classe de test unitaire qui permettra de tester la classe LoginGenerator.
3. Dans la suite de cet exercice, on considère les cas de tests imposés suivants:
   pour chaque cas de test, l'instance de LoginService utilisée est initialisée de la manière suivante :
   ```
   loginService = new LoginService(new String[] {"JROL", "BPER",
   "CGUR", "JDUP", "JRAL", "JRAL1"});
   ```
   1. quand on génère le login de "Paul Durand", on vérifie que le login généré et ajouté à la liste des logins existants est "PDUR" ;
   2. quand on génère le login de "Jean Rolling", on vérifie que le login généré et ajouté à la liste des logins existants est "JROL1" ;
   3. quand on génère le login de "Paul Dùrand", on vérifie que le login généré et ajouté à la liste des logins existants est "PDUR".
4. Commencez par coder un seul de ces tests, puis commitez (sans pousser).
5. Vous-même ou l'un de vos collaborateurs remarque(z) qu'il y a un bug dans le code de LoginGenerator : lorsqu'un login existe déjà, le suffixe ajouté est toujours 1, alors qu'il devrait être incrémenté si nécessaire… Créez une issue GitHub (#3) munie du label « bug » pour mémoriser l'existence du bug, et assignez-vous cette issue.
6. Comme ce bug est directement lié à la classe LoginGenerator que vous êtes en train de tester, vous décidez de rester sur la même branche testsLoginGenerator, tout en prévoyant de mentionner l'issue #3 dans le commit correspondant :
   1. Ajoutez un test unitaire basé sur le cas de test suivant : quand on génère le login de "John Ralling", on vérifie que le login généré et ajouté à la liste des logins existants est "JRAL2".
   2. Après avoir constaté que ce test ne passe pas, modifiez le code de la classe LoginGenerator afin que vos tests passent.
   3. Une fois le bug corrigé, commitez en ajoutant « fix #3 » au message de commit.
7. Toujours dans la même branche, poursuivez le travail de couverture du code par les tests en implémentant les cas de tests ci-dessus avec un ou plusieurs commits.
8. Lorsque vous êtes arrivés à 100 % de couverture par les tests, effectuez un « rebase --interactive » à partir du début de la branche pour « nettoyer » votre historique avant de pousser votre branche locale. Vous pouvez effectuer cette réécriture d'historique en ligne de commande (voir cours) ou via IntelliJ. Ici, nous vous demandons de regrouper tous vos commits effectués dans la branche « testsLoginGenerator » en 1 seul commit. Faites en sorte que le mot-clé « fix #3 » soit conservé au sein du message de commit.

9. Poussez votre branche locale « testsLoginGenerator ».
10. Connectez-vous sur GiHub et ouvrez une demande de Pull Request pour fusionner « testsLoginGenerator » dans « master ».
11. Explorez la page du Pull Request ainsi obtenu (normalement numéroté #4) : vous pouvez p.ex. poster un message ou ajouter un commentaire à l'une des lignes de code.
12. Acceptez la demande de fusion sur GitHub en cliquant sur « Merge Pull Request » avec un message de commit contenant le mot-clé « close #2 ». Enfin, supprimez les « feature branches » locales et distantes ayant été mergées (ici, testsLoginGenerator).
    Remarque : le merge du Pull Request est en principe fait par un autre développeur de l'équipe après la phase de revue de code et les corrections éventuelles.
13. Récupérez localement la dernière version de master et affichez l'historique obtenu.